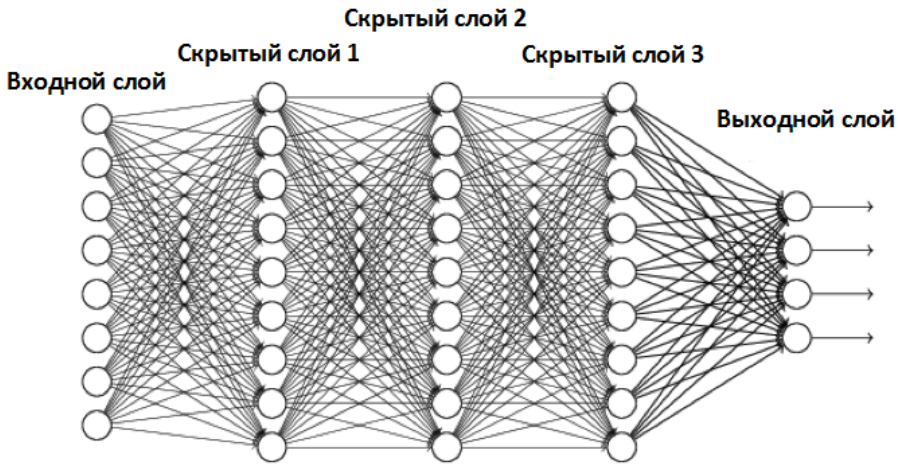
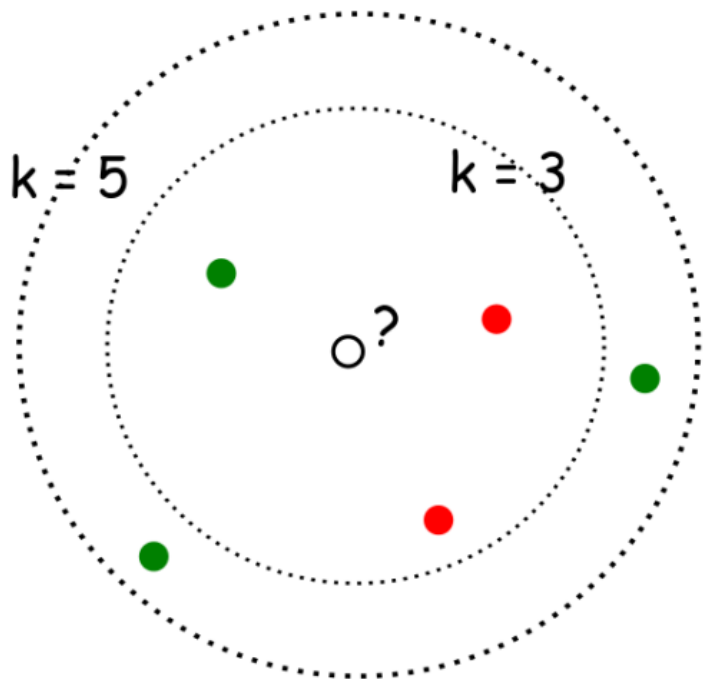


Построение и улучшение алгоритма K-ближайших соседей в Python

Дата публикации Mar 28, 2018

Алгоритм K-Nearest Neighbors, сокращенно K-NN, является классическим алгоритмом работы с машинным обучением, который часто игнорируется в день глубокого обучения. В этом руководстве мы создадим алгоритм K-NN в Scikit-Learn и запустим его в наборе данных MNIST. Оттуда мы создадим наш собственный алгоритм K-NN в надежде на разработку классификатора с большей точностью и скоростью классификации, чем в Scikit-Learn K-NN.

K-модель классификации ближайших соседей



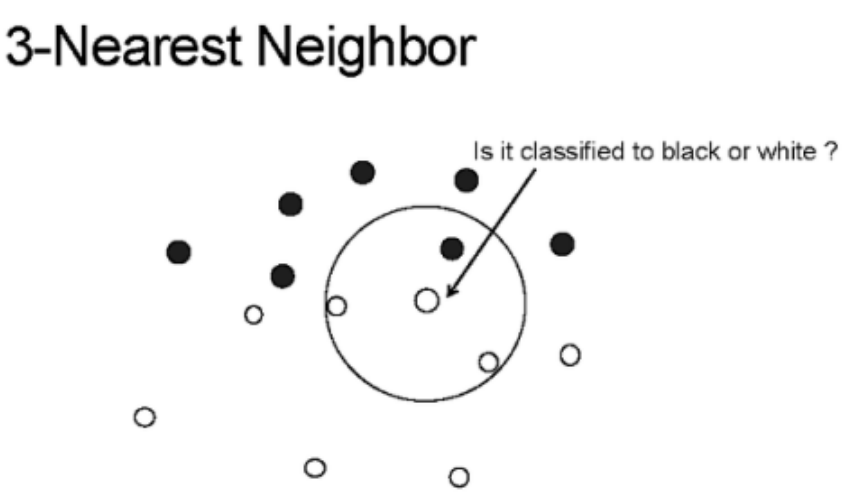
Ленивый Программист

Алгоритм K-Nearest Neighbours представляет собой контролируемый алгоритм машинного обучения, который прост в реализации и в то же время имеет возможность создавать надежные классификации. Одним из самых больших преимуществ K-NN является то, что это*ленивая обучаемость*, Это означает, что модель не требует обучения и может получить право на классификацию данных, в отличие от других родственных им элементов ML, таких как SVM, регрессия и многоуровневое восприятие.

Как работает K-NN

Чтобы классифицировать некоторую заданную точку данных,**п**модель K-NN будет сначала сравниватьпв любой другой момент он имеет в своей базе данных, используя некоторые*метрика расстояния*, Метрика расстояния является чем-то вродеЕвклидово расстояние,простая функция, которая берет две точки и возвращает расстояние между этими двумя точками. Таким образом, можно предположить, что две точки с меньшим расстоянием между ними*больше похоже*чем две точки с большим расстоянием между ними. Это центральная идея K-NN.

Этот процесс вернет неупорядоченный массив, где каждая запись в массиве содержит расстояние между**п**и один из**N**точки данных в базе данных моделей. Таким образом, возвращаемый массив будет иметь размер**N**, Это где K часть ближайших соседей входит:**К**выбрано произвольное значение (обычно между 3–11), которое сообщает модели*сколько**самый**аналогичные пункты***п**это следует учитывать при классификации**п**, Затем модель примет те**К**наиболее близкие значения, и использовать технику голосования, чтобы решить, как классифицировать**п**, как показано на рисунке ниже.



Ленивый Программист

Модель K-NN на изображении имеет**К**значение 3, и точка в центре со стрелкой, указывающей на это**п**, точка, которая должна быть классифицирована. Как вы можете видеть, три точки в круге - это три точки, наиболее близкие или наиболее похожие на**п**, Таким образом, используя простую технику голосования,**п**будет классифицироваться как «белый», так как белый составляет большинство**К**самые похожие значения.

Довольно круто! Удивительно, но этот простой алгоритм может достигать сверхчеловеческих результатов в определенных ситуациях и может быть применен к широкому кругу проблем, как мы увидим далее.

Реализация алгоритма K-NN в Scikit-Learn для классификации изображений MNIST

Данные:

В этом примере мы будем использовать вездесущий набор данных MNIST. Набор данных MNIST является одним из наиболее распространенных наборов данных, используемых в машинном обучении, поскольку его легко реализовать, но он служит надежным методом для проверки моделей.

MNIST - это набор данных из 70000 рукописных цифр, пронумерованных от 0 до 9. Нет двух одинаковых цифр, написанных от руки, и некоторые из них могут быть очень трудно правильно классифицировать. Человеческий эталон для классификации MNIST - точность около 97,5%, поэтому наша цель - победить это!

Алгоритм:

Мы будем использовать`KNeighborsClassifier()`из библиотеки Scikit-Learn Python для запуска. Эта функция принимает много аргументов, но нам нужно будет беспокоиться только о нескольких в этом примере. В частности, мы будем передавать только значение для`n_neighbors`аргумент (это**K**ценность).`weights`Аргумент дает тип системы голосования, используемой моделью, где значением по умолчанию является`uniform`, означая каждый из**K**баллы одинаково взвешены при классификации**п**,`algorithm`Аргумент также останется со значением по умолчанию`auto`, так как мы хотим, чтобы Scikit-Learn нашла оптимальный алгоритм для классификации самих данных MNIST.

Ниже я встраиваю блокнот Jupyter, который создает классификатор K-NN с помощью Scikit-Learn. Вот так!

```
action 'view' is not supported for filetype 'ipynb'
```

SKL-KNN.ipynb hosted with ❤ by GitHub view raw

Фантастика! Мы создали очень простую модель K-ближайших соседей, используя Scikit-Learn, которая достигла необычайной производительности в наборе данных MNIST.

Проблема? Ну, это заняло много времени, чтобы классифицировать эти точки (8 минут и почти 4 минуты, соответственно, для двух наборов данных), и по иронии судьбы K-NN по-прежнему является одним из самых быстрых методов классификации. Должен быть более быстрый путь ...

Построение более быстрой модели

Большинство моделей K-NN используют евклидово или манхэттенское расстояние в качестве показателя расстояния до точки. Эти показатели просты и хорошо работают в самых разных ситуациях.

Одна метрика расстояния, которая используется редкокосинус сходство, Косинусное сходство, как правило, не является метрикой расстояния, так как оно нарушаетнеравенство треугольника,и не работает с отрицательными данными. Однако косинусное сходство идеально подходит для MNIST. Это быстро, просто и получает немного лучшую точность, чем другие метрики расстояния в MNIST Но чтобы действительно добиться максимальной производительности, нам нужно написать собственную модель K-NN. После того, как мы самостоятельно создали модель K-NN, мы должны получить более высокую производительность, чем модель Scikit-Learn, и, возможно, даже более высокую точность. Давайте посмотрим на ноутбук ниже, где мы создаем нашу собственную модель K-NN.

```
action 'view' is not supported for filetype 'ipynb'
```

COS-KNN.ipynb hosted with ❤ by GitHub view raw

Как показано в записной книжке, модель K-NN, которую мы создали, превосходит Scikit-Learn K-NN с точки зрения как скорости классификации (со значительным запасом), так и точности (улучшение на 1% для одного набора данных)! Теперь мы можем продолжить реализацию этой модели на практике, зная, что мы разработали действительно быстрый алгоритм.

Вывод

Это было много, но мы выучили пару ценных уроков. Сначала мы узнали, как работает K-NN и как его легко реализовать. Но самое главное, мы узнали, что важно всегда учитывать проблему, которую вы пытаетесь решить, и инструменты, которые у вас есть для ее решения. Время от времени лучше всего тратить время на эксперименты - и да, на создание собственных моделей - при решении проблемы. Как доказано в ноутбуках, он может приносить огромные дивиденды: наша вторая запатентованная модель позволила ускорить работу в 1,5–2 раза, сэкономив объекту много времени.

Если вы хотите узнать больше, я призываю вас оформить заказэтот репозиторий GitHub, где вы найдете более тщательный анализ между двумя моделями и некоторые более интересные особенности о нашей более быстрой модели K-NN!

Пожалуйста, оставляйте любые комментарии, критику или вопросы в комментариях!

Оригинальная статья

- [Фреймворки и библиотеки (большая подборка ссылок для разных языков программирования)]
- [Список бесплатных книг по машинному обучению, доступных для скачивания]
- [Список блогов и информационных бюллетеней по науке о данных и машинному обучению]
- [Список (в основном) бесплатных курсов машинного обучения, доступных в Интернете]